sampledoc Documentation

Release 1.0

John Hunter, Fernando Perez, Michael Droettboom

August 11, 2009

CONTENTS

This is a tutorial introduction to quickly get you up and running with your own sphinx documentation system. We'll cover everything from installing sphinx, to customizing the look and feel, to using custom extensions for embedding plots, inheritance diagrams, syntax highlighted ipython sessions and more. If you follow along the tutorial, you'll start with nothing and end up with this site – it's the bootstrapping documentation tutorial that writes itself!

The source code for this tutorial lives in mpl svn (see *Fetching the data*) and the sampledoc PDF

Contents:

CHAPTER

ONE

GETTING STARTED

1.1 Installing your doc directory

You may already have sphinx sphinx installed – you can check by doing:

```
python -c 'import sphinx'
```

If that fails grab the latest version of and install it with:

```
> sudo easy_install sphinx
```

Now you are ready to build a template for your docs, using sphinx-quickstart:

```
> sphinx-quickstart
```

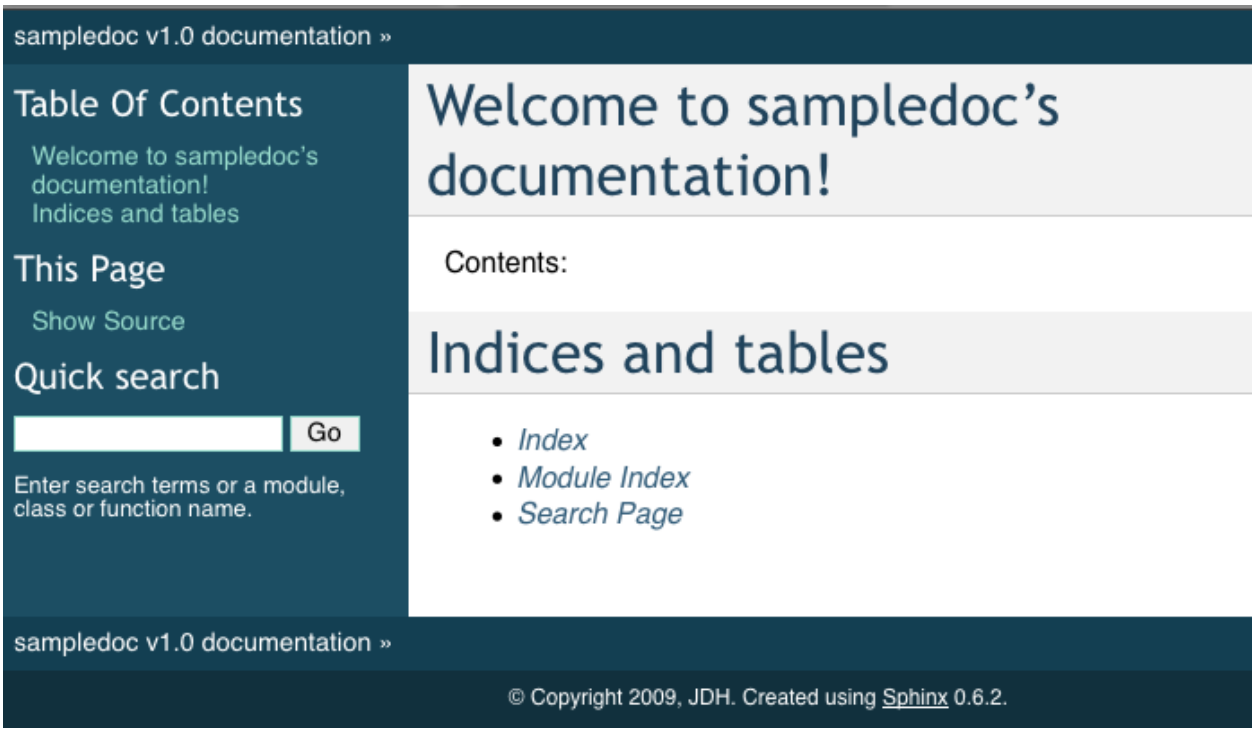
accepting most of the defaults. I choose "sampledoc" as the name of my project. cd into your new directory and check the contents:

```
home:~/tmp/sampledoc> ls
Makefile      _static         conf.py
_build                _templates      index.rst
```

The index.rst is the master ReST for your project, but before adding anything, let's see if we can build some html:

```
make html
```

If you now point your browser to `_build/html/index.html`, you should see a basic sphinx site.



1.1.1 Fetching the data

Now we will start to customize out docs. Grab a couple of files from the web site or svn. You will need `getting_started.rst` and `_static/basic_screenshot.png`. All of the files live in the "completed" version of this tutorial, but since this is a tutorial, we'll just grab them one at a time, so you can learn what needs to be changed where. Since we have more files to come, I'm going to grab the whole svn directory and just copy the files I need over for now. First, I'll cd to the directory containing my project, and get the "finished" product, and then copy in just the files I need:

```
home:~/tmp/sampledoc> pwd
/Users/jdhunter/tmp/sampledoc
home:~/tmp/sampledoc> cd ..
home:~/tmp> svn co https://matplotlib.svn.sourceforge.net/svnroot/\
matplotlib/trunk/sampledoc_tut
A    sampledoc_tut/cheatsheet.rst
A    sampledoc_tut/_static
A    sampledoc_tut/_static/basic_screenshot.png
A    sampledoc_tut/conf.py
A    sampledoc_tut/Makefile
A    sampledoc_tut/_templates
A    sampledoc_tut/_build
A    sampledoc_tut/getting_started.rst
A    sampledoc_tut/index.rst
Checked out revision 7449.
home:~/tmp> cp sampledoc_tut/getting_started.rst sampledoc/
home:~/tmp> cp sampledoc_tut/_static/basic_screenshot.png \
sampledoc/_static/
```

Now we are ready to rebuild the docs. We used the image directory to include to the screenshot above with:

```
.. image:: _static/basic_screenshot.png
```

The last step is to modify `index.rst` to include the getting_started file (be careful with the indentation, the "getting_started" should line up with the ':' in `:maxdepth`:

```
Contents:

.. toctree::
   :maxdepth: 2

   getting_started.rst
```

and then rebuild the docs with `make html`. When you reload the page, you should see a link to the "Getting Started" docs, and in there this page with the screenshot. *Voila!*

Next we'll customize the look and feel of our site to give it a logo, some custom css, and update the navigation panels to look more like the sphinx site itself – see *Customizing the look and feel of the site*.

CHAPTER

TWO

CUSTOMIZING THE LOOK AND FEEL OF THE SITE

The sphinx site itself looks better than the sites created with the default css, so here we'll invoke Tufte's phrase "Intelligence imitates but genious steals and grab their css and part of their layout. As before, you can either get the required files `_static/default.css`, `_templates:layout.html` and `_staticlogo.png` from the website or svn (see *Fetching the data*). Since I did a svn checkout before, I will just copy the stuff I need from there:

```
home:~/tmp/sampledoc> cp ../sampledoc_tut/_static/default.css _static/
home:~/tmp/sampledoc> cp ../sampledoc_tut/_templates/layout.html _templates/
home:~/tmp/sampledoc> cp ../sampledoc_tut/_static/logo.png _static/
home:~/tmp/sampledoc> ls _static/ _templates/
_static/:
basic_screenshot.png        default.css             logo.png

_templates/:
layout.html
```

Sphinx will automatically pick up the css and layout html files since we put them in the default places with the default names, but we have to manually include the logo in our `layout.html`. Let's take a look at the layout file: the first part puts a horizontal navigation bar at the top of our page, like you see on the sphinx and matplotlib sites, the second part includes a logo that when we click on it will take us *home* and the last part moves the vertical navigation panels to the right side of the page:

```
{% extends "!layout.html" %}


{% block rootrellink %}
        <li><a href="{{ pathto('index') }}">home</a>| </li>
        <li><a href="{{ pathto('search') }}">search</a>| </li>
       <li><a href="{{ pathto('contents') }}">documentation </a> &raquo;</li>
{% endblock %}


{% block relbar1 %}

<div style="background-color: white; text-align: left; padding: 10px 10px 15px 15px">
<a href="{{ pathto('index') }}"><img src="{{
pathto("_static/logo.png", 1) }}" border="0" alt="sampledoc"/></a>
</div>
{{ super() }}
{% endblock %}
```

```
{# put the sidebar before the body #}
{% block sidebar1 %}{{ sidebar() }}{% endblock %}
{% block sidebar2 %}{% endblock %}
```

Once you rebuild the site with a `make html` and reload the page in your browser, you should see a fancier site that looks like this

CHAPTER

THREE

SPHINX EXTENSIONS FOR EMBEDDED PLOTS, MATH AND MORE

Sphinx is written in python, and supports the ability to write custom extensions. We've written a few for the matplotlib documentation, some of which are part of matplotlib itself in the matplotlib.sphinxext module, some of which are included only in the sphinx doc directory, and there are other extensions written by other groups, eg numpy and ipython. We're collecting these in this tutorial and showing you how to install and use them for your own project. First let's grab the python extension files from the `sphinxext` directory from svn (see *Fetching the data*, and install them in our `sampledoc` project `sphinxext` directory:

```
home:~/tmp/sampledoc> mkdir sphinxext
home:~/tmp/sampledoc> cp ../sampledoc_tut/sphinxext/*.py sphinxext/
home:~/tmp/sampledoc> ls sphinxext/
apigen.py              inheritance_diagram.py
docscrape.py           ipython_console_highlighting.py
docscrape_sphinx.py    numpydoc.py
```

In addition to the builtin matplotlib extensions for embedding pyplot plots and rendering math with matplotlib's native math engine, we also have extensions for syntax highlighting ipython sessions, making inhertiance diagrams, and more.

We need to inform sphinx of our new extensions in the `conf.py` file by adding the following. First we tell it where to find the extensions:

```
# If your extensions are in another directory, add it here. If the
# directory is relative to the documentation root, use
# os.path.abspath to make it absolute, like shown here.
sys.path.append(os.path.abspath('sphinxext'))
```

And then we tell it what extensions to load:

```
# Add any Sphinx extension module names here, as strings. They can
# be extensions coming with Sphinx (named 'sphinx.ext.*') or your
# custom ones.
extensions = ['matplotlib.sphinxext.mathmpl',
          'matplotlib.sphinxext.only_directives',
          'matplotlib.sphinxext.plot_directive',
          'sphinx.ext.autodoc',
          'sphinx.ext.doctest',
          'ipython_console_highlighting',
          'inheritance_diagram',
          'numpydoc']
```

Now let's look at some of these in action. You can see the literal source for this file at *This file*.

3.1 ipython sessions

Michael Droettboom contributed a sphinx extension which does pygments syntax highlighting on ipython sessions. Just use ipython as the language in the sourcecode directive:

```
.. sourcecode:: ipython

    In [69]: lines = plot([1,2,3])

    In [70]: setp(lines)
      alpha: float
      animated: [True | False]
      antialiased or aa: [True | False]
      ...snip
```

and you will get the syntax highlighted output below.

```
In [69]: lines = plot([1,2,3])

In [70]: setp(lines)
  alpha: float
  animated: [True | False]
  antialiased or aa: [True | False]
  ...snip
```

This support is included in this template, but will also be included in a future version of Pygments by default.

3.2 Using math

In sphinx you can include inline math $x \leftarrow y\ x \forall y\ x - y$ or display math

$$W^{3\beta}_{\delta_1 \rho_1 \sigma_2} = U^{3\beta}_{\delta_1 \rho_1} + \frac{1}{8\pi 2} \int_{\alpha_2}^{\alpha_2} d\alpha_2' \left[\frac{U^{2\beta}_{\delta_1 \rho_1} - \alpha_2' U^{1\beta}_{\rho_1 \sigma_2}}{U^{0\beta}_{\rho_1 \sigma_2}} \right] \tag{3.1}$$

To include math in your document, just use the math directive; here is a simpler equation:

```
.. math::

  W^{3\beta}_{\delta_1 \rho_1 \sigma_2} \approx U^{3\beta}_{\delta_1 \rho_1}
```

which is rendered as

$$W^{3\beta}_{\delta_1 \rho_1 \sigma_2} \approx U^{3\beta}_{\delta_1 \rho_1} \tag{3.2}$$

This documentation framework includes a Sphinx extension, `sphinxext/mathmpl.py`, that uses matplotlib to render math equations when generating HTML, and LaTeX itself when generating a PDF. This can be useful on systems that have matplotlib, but not LaTeX, installed. To use it, add `mathmpl` to the list of extensions in `conf.py`.

Current SVN versions of Sphinx now include built-in support for math. There are two flavors:

- pngmath: uses dvipng to render the equation
- jsmath: renders the math in the browser using Javascript

To use these extensions instead, add `sphinx.ext.pngmath` or `sphinx.ext.jsmath` to the list of extensions in `conf.py`.

All three of these options for math are designed to behave in the same way.

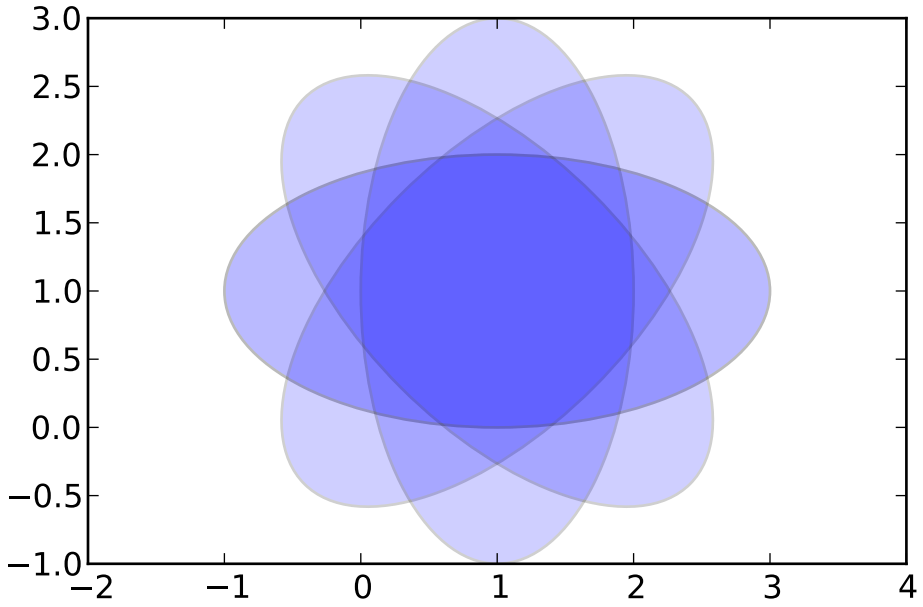
3.3 Inserting matplotlib plots

Inserting automatically-generated plots is easy. Simply put the script to generate the plot in the `pyplots` directory, and refer to it using the `plot` directive. To include the source code for the plot in the document, pass the `include-source` parameter:

```
.. plot:: pyplots/ellipses.py
   :include-source:
```

In the HTML version of the document, the plot includes links to the original source code, a high-resolution PNG and a PDF. In the PDF version of the document, the plot is included as a scalable PDF.

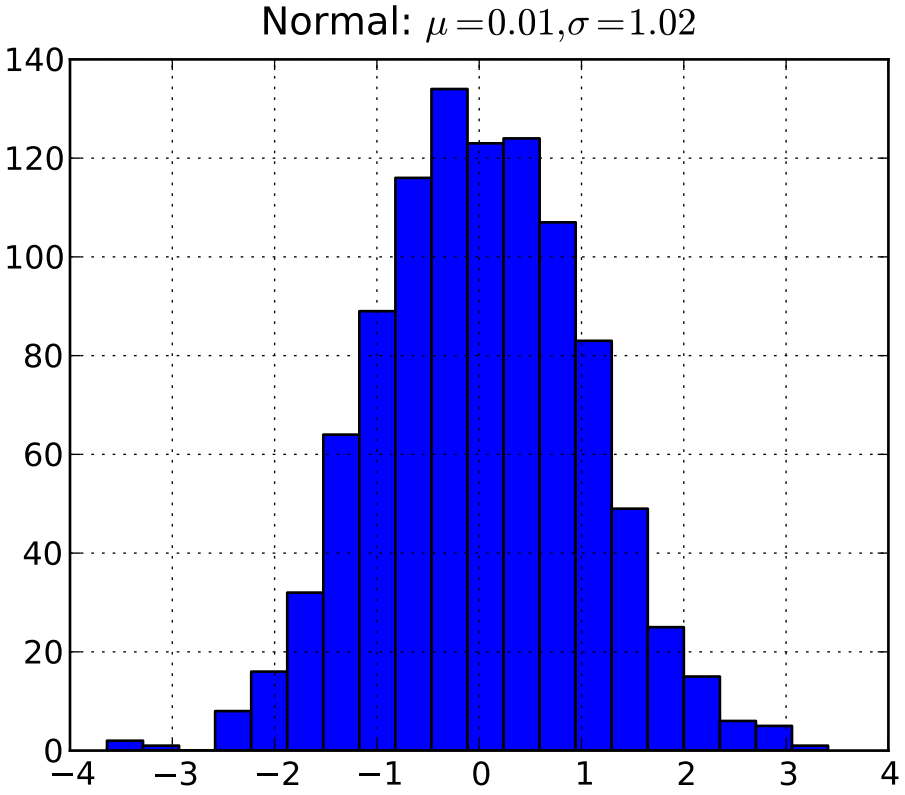
```
from pylab import *
from matplotlib.patches import Ellipse

delta = 45.0 # degrees

angles = arange(0, 360+delta, delta)
ells = [Ellipse((1, 1), 4, 2, a) for a in angles]

a = subplot(111, aspect='equal')

for e in ells:
    e.set_clip_box(a.bbox)
    e.set_alpha(0.1)
    a.add_artist(e)

xlim(-2, 4)
ylim(-1, 3)

show()
```



You can also inline simple plots, and the code will be executed at documentation build time and the figure inserted into your docs; the following code:

```
.. plot::

   import matplotlib.pyplot as plt
   import numpy as np
   x = np.random.randn(1000)
   plt.hist( x, 20)
   plt.grid()
   plt.title(r'Normal: $\mu=%.2f, \sigma=%.2f$'%(x.mean(), x.std()))
   plt.show()
```

produces this output:



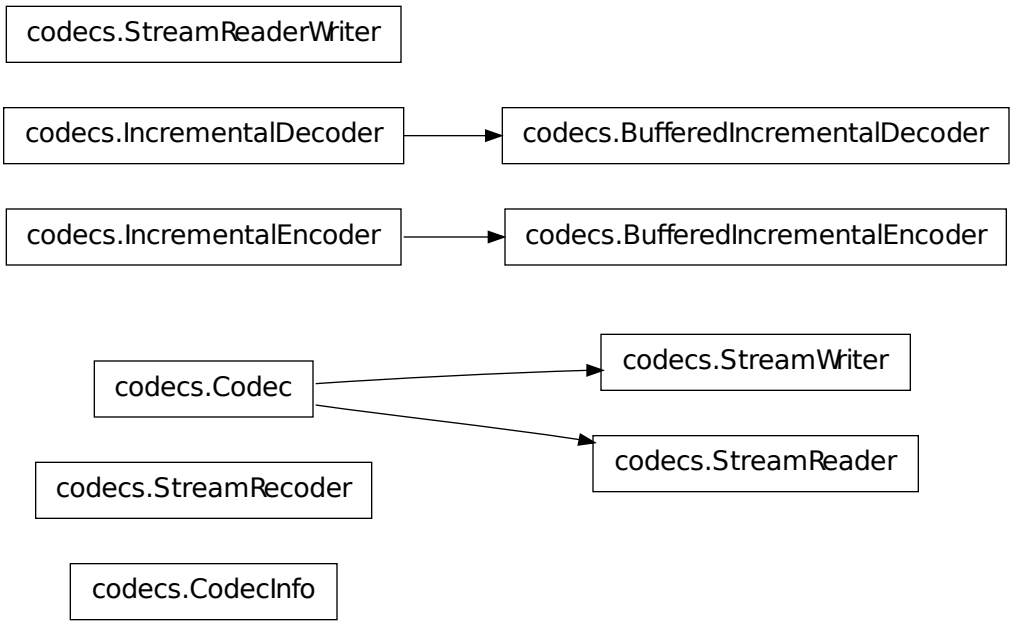
3.4 Inheritance diagrams

Inheritance diagrams can be inserted directly into the document by providing a list of class or module names to the `inheritance-diagram` directive.

For example:

```
.. inheritance-diagram:: codecs
```

produces:



3.5 This file

```
.. _extensions:


*************************************************
Sphinx extensions for embedded plots, math and more
*************************************************

Sphinx is written in python, and supports the ability to write custom
extensions.  We've written a few for the matplotlib documentation,
some of which are part of matplotlib itself in the
matplotlib.sphinxext module, some of which are included only in the
sphinx doc directory, and there are other extensions written by other
groups, eg numpy and ipython.  We're collecting these in this tutorial
and showing you how to install and use them for your own project.
First let's grab the python extension files from the :file:`sphinxext`
directory from svn (see :ref:`fetching-the-data`, and install them in
our :file:`sampledoc` project :file:`sphinxext` directory::

    home:~/tmp/sampledoc> mkdir sphinxext
    home:~/tmp/sampledoc> cp ../sampledoc_tut/sphinxext/*.py sphinxext/
    home:~/tmp/sampledoc> ls sphinxext/
    apigen.py             inheritance_diagram.py
    docscrape.py          ipython_console_highlighting.py
    docscrape_sphinx.py   numpydoc.py
```

```
In addition to the builtin matplotlib extensions for embedding pyplot
plots and rendering math with matplotlib's native math engine, we also
have extensions for syntax highlighting ipython sessions, making
inhertiance diagrams, and more.

We need to inform sphinx of our new extensions in the :file:`conf.py`
file by adding the following.  First we tell it where to find the extensions::

    # If your extensions are in another directory, add it here. If the
    # directory is relative to the documentation root, use
    # os.path.abspath to make it absolute, like shown here.
    sys.path.append(os.path.abspath('sphinxext'))

And then we tell it what extensions to load::

    # Add any Sphinx extension module names here, as strings. They can
    # be extensions coming with Sphinx (named 'sphinx.ext.*') or your
    # custom ones.
    extensions = ['matplotlib.sphinxext.mathmpl',
              'matplotlib.sphinxext.only_directives',
              'matplotlib.sphinxext.plot_directive',
              'sphinx.ext.autodoc',
              'sphinx.ext.doctest',
              'ipython_console_highlighting',
              'inheritance_diagram',
              'numpydoc']


Now let's look at some of these in action.  You can see the literal
source for this file at :ref:`extensions-literal`.


.. _ipython-highlighting:

ipython sessions
================

Michael Droettboom contributed a sphinx extension which does pygments
syntax highlighting on ipython sessions.  Just use ipython as the
language in the sourcecode directive::

    .. sourcecode:: ipython

        In [69]: lines = plot([1,2,3])

        In [70]: setp(lines)
          alpha: float
          animated: [True | False]
          antialiased or aa: [True | False]
          ...snip


and you will get the syntax highlighted output below.

.. sourcecode:: ipython

    In [69]: lines = plot([1,2,3])
```

```
    In [70]: setp(lines)
      alpha: float
      animated: [True | False]
      antialiased or aa: [True | False]
      ...snip

This support is included in this template, but will also be included
in a future version of Pygments by default.

.. _using-math:

Using math
==========

In sphinx you can include inline math :math:`x\leftarrow y\ x\forall
y\ x-y` or display math

.. math::

  W^{3\beta}_{\delta_1 \rho_1 \sigma_2} = U^{3\beta}_{\delta_1 \rho_1} + \frac{1}{8 \pi 2} \int^{\alp

To include math in your document, just use the math directive; here is
a simpler equation::

    .. math::

      W^{3\beta}_{\delta_1 \rho_1 \sigma_2} \approx U^{3\beta}_{\delta_1 \rho_1}

which is rendered as

.. math::

   W^{3\beta}_{\delta_1 \rho_1 \sigma_2} \approx U^{3\beta}_{\delta_1 \rho_1}

This documentation framework includes a Sphinx extension,
:file:`sphinxext/mathmpl.py`, that uses matplotlib to render math
equations when generating HTML, and LaTeX itself when generating a
PDF.  This can be useful on systems that have matplotlib, but not
LaTeX, installed.  To use it, add ``mathmpl`` to the list of
extensions in :file:`conf.py`.

Current SVN versions of Sphinx now include built-in support for math.
There are two flavors:

  - pngmath: uses dvipng to render the equation

  - jsmath: renders the math in the browser using Javascript

To use these extensions instead, add ``sphinx.ext.pngmath`` or
``sphinx.ext.jsmath`` to the list of extensions in :file:`conf.py`.

All three of these options for math are designed to behave in the same
way.

.. _pyplots:

Inserting matplotlib plots
==========================
```

```
Inserting automatically-generated plots is easy.  Simply put the
script to generate the plot in the :file:`pyplots` directory, and
refer to it using the ``plot`` directive.  To include the source code
for the plot in the document, pass the ``include-source`` parameter::

  .. plot:: pyplots/ellipses.py
     :include-source:

In the HTML version of the document, the plot includes links to the
original source code, a high-resolution PNG and a PDF.  In the PDF
version of the document, the plot is included as a scalable PDF.

.. plot:: pyplots/ellipses.py
   :include-source:


You can also inline simple plots, and the code will be executed at
documentation build time and the figure inserted into your docs; the
following code::

   .. plot::

      import matplotlib.pyplot as plt
      import numpy as np
      x = np.random.randn(1000)
      plt.hist( x, 20)
      plt.grid()
      plt.title(r'Normal: $\mu=%.2f, \sigma=%.2f$'%(x.mean(), x.std()))
      plt.show()

produces this output:

.. plot::

    import matplotlib.pyplot as plt
    import numpy as np
    x = np.random.randn(1000)
    plt.hist( x, 20)
    plt.grid()
    plt.title(r'Normal: $\mu=%.2f, \sigma=%.2f$'%(x.mean(), x.std()))
    plt.show()



Inheritance diagrams
====================

Inheritance diagrams can be inserted directly into the document by
providing a list of class or module names to the
``inheritance-diagram`` directive.

For example::

  .. inheritance-diagram:: codecs

produces:
```

```
.. inheritance-diagram:: codecs


.. _extensions-literal:

This file
=========

.. literalinclude:: extensions.rst
```

CHAPTER

FOUR

SPHINX CHEAT SHEET

Here is a quick and dirty cheat sheet for some common stuff you want to do in sphinx and ReST. You can see the literal source for this file at *This file*.

4.1 Formatting text

You use inline markup to make text *italics*, **bold**, or `monotype`.

You can represent code blocks fairly easily:

```
import numpy as np
x = np.random.rand(12)
```

Or literally include code:

```
from pylab import *
from matplotlib.patches import Ellipse

delta = 45.0 # degrees

angles = arange(0, 360+delta, delta)
ells = [Ellipse((1, 1), 4, 2, a) for a in angles]

a = subplot(111, aspect='equal')

for e in ells:
    e.set_clip_box(a.bbox)
    e.set_alpha(0.1)
    a.add_artist(e)

xlim(-2, 4)
ylim(-1, 3)

show()
```

4.2 Making a list

It is easy to make lists in rest

4.2.1 Bullet points

This is a subsection making bullet points

- point A
- point B
- point C

4.2.2 Enumerated points

This is a subsection making numbered points

1. point A
2. point B
3. point C

4.3 Making a table

This shows you how to make a table – if you only want to make a list see *Making a list*.

Name	Age
John D Hunter	40
Cast of Thousands	41
And Still More	42

4.4 Making links

It is easy to make a link to yahoo or to some section inside this document (see *Making a table*) or another document.

You can also reference classes, modules, functions, etc that are documented using the sphinx autodoc facilites. For example, see the module `matplotlib.backend_bases` documentation, or the class `LocationEvent`, or the method `mpl_connect()`.

4.5 This file

```
.. _cheat-sheet:

******************
Sphinx cheat sheet
******************

Here is a quick and dirty cheat sheet for some common stuff you want
to do in sphinx and ReST.  You can see the literal source for this
file at :ref:`cheatsheet
-literal`.


.. _formatting-text:
```

```
Formatting text
===============

You use inline markup to make text *italics*, **bold**, or ``monotype``.

You can represent code blocks fairly easily::

   import numpy as np
   x = np.random.rand(12)

Or literally include code:

.. literalinclude:: pyplots/ellipses.py

.. _making-a-list:

Making a list
=============

It is easy to make lists in rest

Bullet points
-------------

This is a subsection making bullet points

* point A

* point B

* point C


Enumerated points
-----------------

This is a subsection making numbered points

#. point A

#. point B

#. point C


.. _making-a-table:

Making a table
==============

This shows you how to make a table -- if you only want to make a list see :ref:`making-a-list`.

==================   ============
Name                 Age
==================   ============
John D Hunter        40
Cast of Thousands    41
```

```
And Still More       42
==================   ============

.. _making-links:

Making links
============

It is easy to make a link to `yahoo <http://yahoo.com>`_ or to some
section inside this document (see :ref:`making-a-table`) or another
document.

You can also reference classes, modules, functions, etc that are
documented using the sphinx `autodoc
<http://sphinx.pocoo.org/ext/autodoc.html>`_ facilites.  For example,
see the module :mod:`matplotlib.backend_bases` documentation, or the
class :class:`~matplotlib.backend_bases.LocationEvent`, or the method
:meth:`~matplotlib.backend_bases.FigureCanvasBase.mpl_connect`.



.. _cheatsheet-literal:

This file
=========

.. literalinclude:: cheatsheet.rst
```

CHAPTER

FIVE

EMACS REST SUPPORT

5.1 Emacs helpers

There is an emacs mode rst.el which automates many important ReST tasks like building and updateing table-of-contents, and promoting or demoting section headings. Here is the basic `.emacs` configuration:

```
(require 'rst)
(setq auto-mode-alist
      (append '(("\\.txt$" . rst-mode)
                ("\\.rst$" . rst-mode)
                ("\\.rest$" . rst-mode)) auto-mode-alist))
```

Some helpful functions:

```
C-c TAB - rst-toc-insert

  Insert table of contents at point

C-c C-u - rst-toc-update

    Update the table of contents at point

C-c C-l rst-shift-region-left

    Shift region to the left

C-c C-r rst-shift-region-right

    Shift region to the right
```

CHAPTER

SIX

INDICES AND TABLES

- *Index*
- *Module Index*
- *Search Page*